

Rendering Plots With Matplotlib

CSE/IT 107L

NMT Department of Computer Science and Engineering

“The greatest value of a picture is when it forces us to notice what we never expected to see.”

—John Tukey

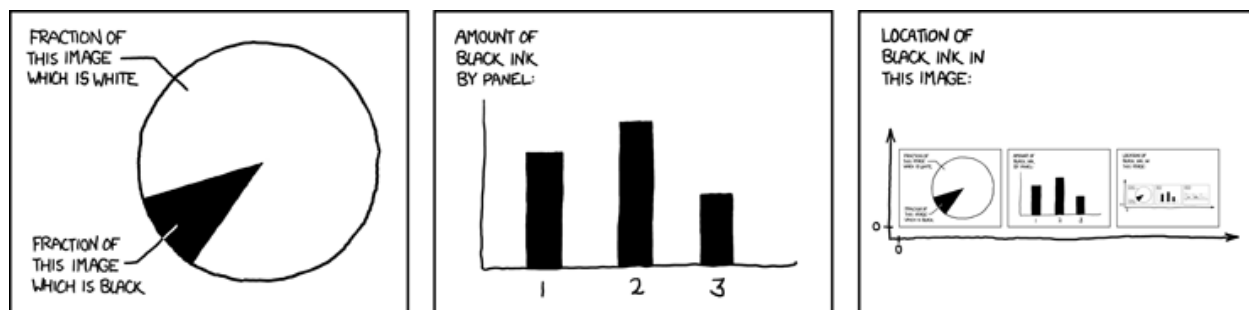


Figure 1: Self description. <https://xkcd.com/688/>

Contents

Introduction	ii
1 Keyword Arguments	1
2 Matplotlib	1
2.1 Plotting Points with the plot Function	2
2.2 Histograms using the hist Function	3
2.3 Scatter plots with scatter Function	5
2.4 Bar Plots with the bar Function	6
2.5 Drawing Many Plots with the subplot Function	7
2.6 Describing your Plots	8
2.7 Subplots and Annotations	11
2.8 Overlaying Plots	12
2.9 Saving Plots to a File Using plt.savefig	13
3 Exploring APIs	13
3.1 Other Libraries	14
4 Exercises	15
5 The CSV Module	17
Submitting	20

Introduction

In this lab you will learn about generating 2D graphics using Matplotlib. This is a powerful Python library that can be used to create detailed visualizations such as scatter plots and histograms. You will learn about Python’s list comprehensions, which help when modifying lists of data. There’s also an extra credit where you use the built-in CSV module to read CSV files. The key lesson this lab should teach you is to read Python library documentation. With this skill you can learn to use many of Python’s free and open-source libraries.

1 Keyword Arguments

Before we start, you should know about a commonly used Python feature called keyword arguments. As you know, functions may take zero or more required arguments, these are called *positional arguments*. There is a special syntax that can be used to define *optional arguments*.

For example, this function takes either one or two arguments. The second argument has a default value of 5.

```
1 >>> def f(fst, snd = 5):
2 ...     return fst * snd
3 >>> f(10)
4 50
5 >>> f(10, 6)
6 60
7 >>> f()
8 Traceback (most recent call last):
9   File "<stdin>", line 1, in <module>
10 TypeError: f() missing 1 required positional argument: 'fst'
11 Traceback (most recent call last):
12   File "<stdin>", line 1, in <module>
13 TypeError: f() missing 1 required positional argument: 'fst'
```

An important feature is that keyword arguments may be labeled and listed in any order when calling the function. For example, the following function takes two keyword arguments. There are five valid ways to call this function. You may pass no arguments, then the default values are used. You may pass either argument by itself or you may pass both arguments in any order.

```
1 >>> def p(greeting="Hello", subject="world"):
2 ...     return "{} {}".format(greeting, subject)
3 >>> p()
4 "Hello world"
5 >>> p(greeting="Goodbye")
6 "Goodbye world"
7 >>> p(subject="humans")
8 "Hello humans"
9 >>> p(greeting="Bonjour", subject="le monde")
10 "Bonjour le monde"
11 >>> p(subject="le monde", greeting="Bonjour")
12 "Bonjour le monde"
```

2 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of formats and interactive environments. We will be using the `matplotlib.pyplot` library. It is commonly renamed to `plt` using this code:

```
1 >>> import matplotlib.pyplot as plt
```

It's a good idea to keep the Matplotlib documentation on hand as you read these examples. All of the following matplotlib functions are documented online at:

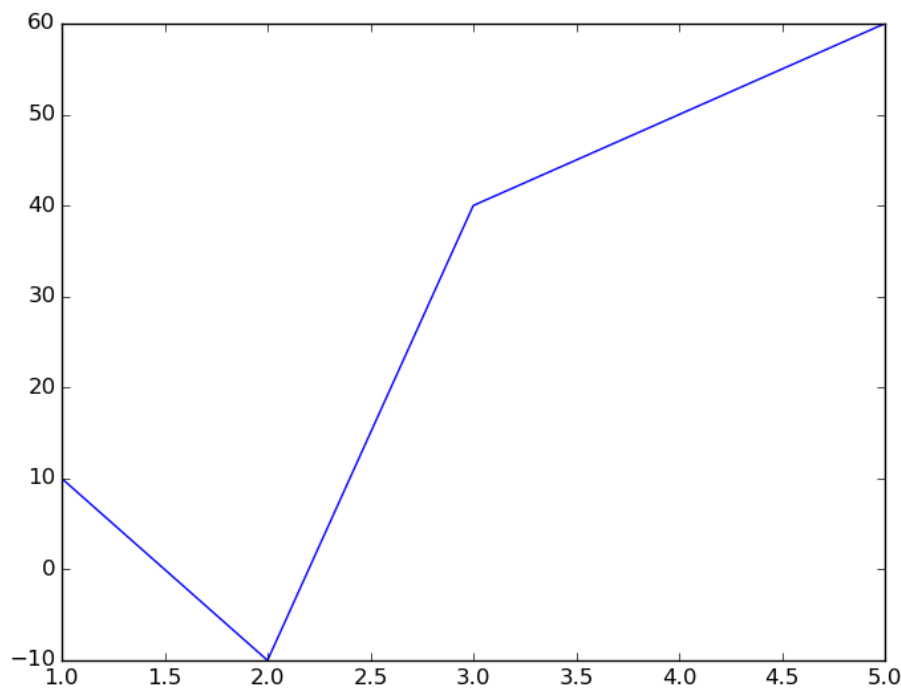
http://matplotlib.org/api/pyplot_api.html

2.1 Plotting Points with the plot Function

`plt.plot` is a versatile function that is used to plot pairs of numeric values. Matplotlib does not take a list of tuples representing points, instead it takes two lists. The list of values on the x-axis are the first argument and the list of y-axis values form the second argument. One of the lists must be in ascending order.

```
1 >>> import matplotlib.pyplot as plt
2 >>> plt.plot([1, 2, 3, 4], [10, -10, 40, 50])
3 >>> plt.show()
```

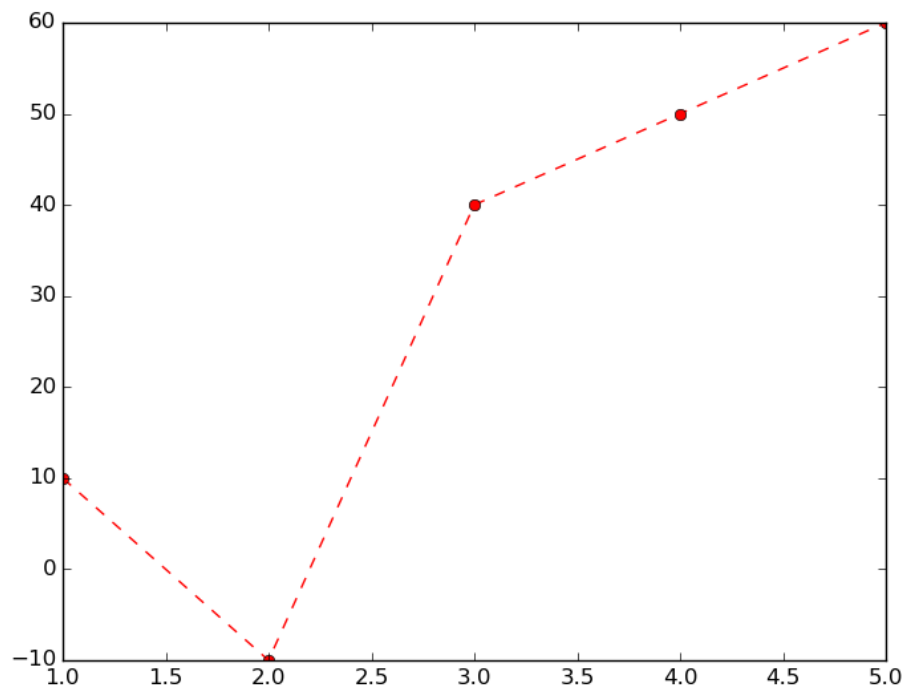
The `plt.show()` function shows everything drawn by previously called plotting functions. This image was drawn by the previous Matplotlib code:



Matplotlib plots can be modified in many ways. The documentation has a table of keyword arguments for this function. The columns are Property and Description. Let's try some of those. The `color`, `linestyle`, and `marker` keyword arguments can be used to draw a differently styled line. As of March, 2016 the Matplotlib documentation for these functions, colors, and line styles are located at

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot
http://matplotlib.org/api/colors_api.html
http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D.set_linestyle

Here's the same plot with a red, dashed line and o's on every data point.

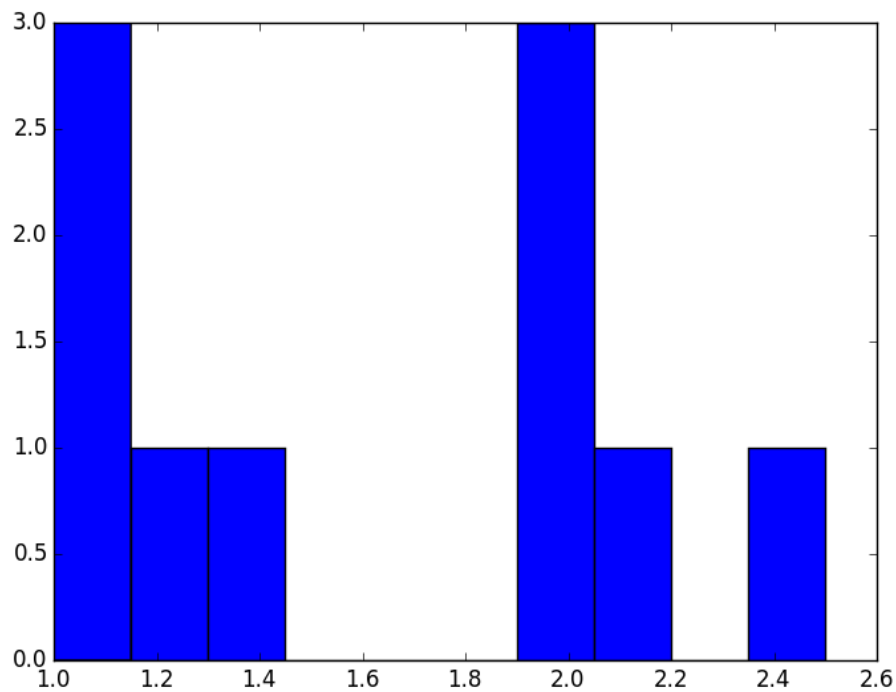


```
1 >>> import matplotlib.pyplot as plt
2 >>> plt.plot([1, 2, 3, 4, 5], [10, -10, 40, 50, 60],
3 ...         color='red', linestyle='dashed', marker='o')
4 >>> plt.show()
```

2.2 Histograms using the hist Function

Matplotlib's `plt.hist()` can draw the histogram of a list, which shows how often a value occurs.

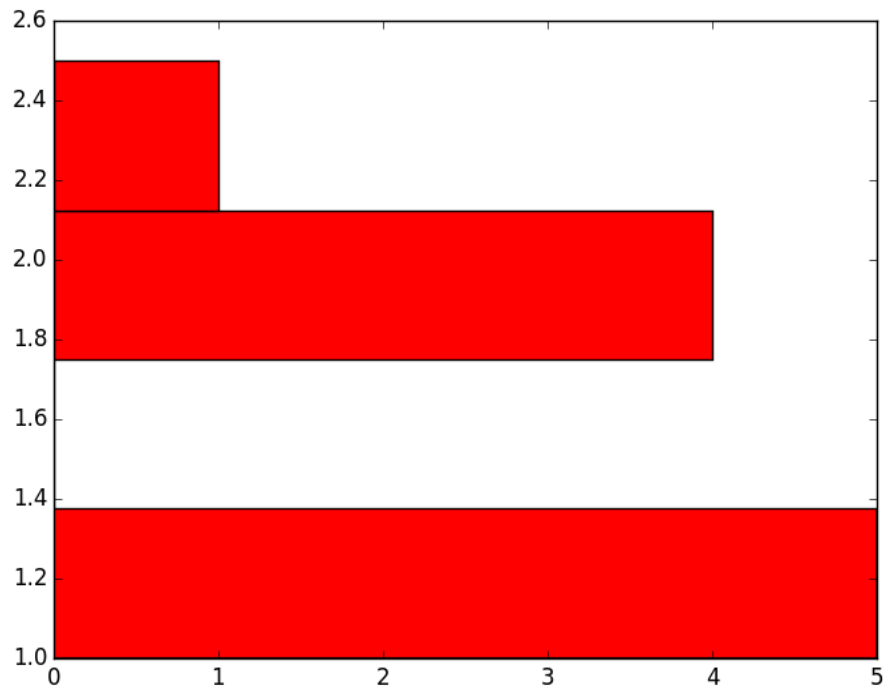
```
1 >>> import matplotlib.pyplot as plt
2 >>> plt.hist([1, 1, 1, 1.2, 1.3, 1.9, 1.9, 2, 2.1, 2.5])
3 >>> plt.show()
```



The `bins` argument is used to draw fewer bins of data. The histogram is rotated by passing `orientation='horizontal'`. See the documentation at:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist

```
1 >>> import matplotlib.pyplot as plt
2 >>> plt.hist([1, 1, 1, 1.2, 1.3, 1.9, 1.9, 2, 2.1, 2.5]),
3 ...         bins=4, orientation='horizontal', color='red')
4 >>> plt.show()
```



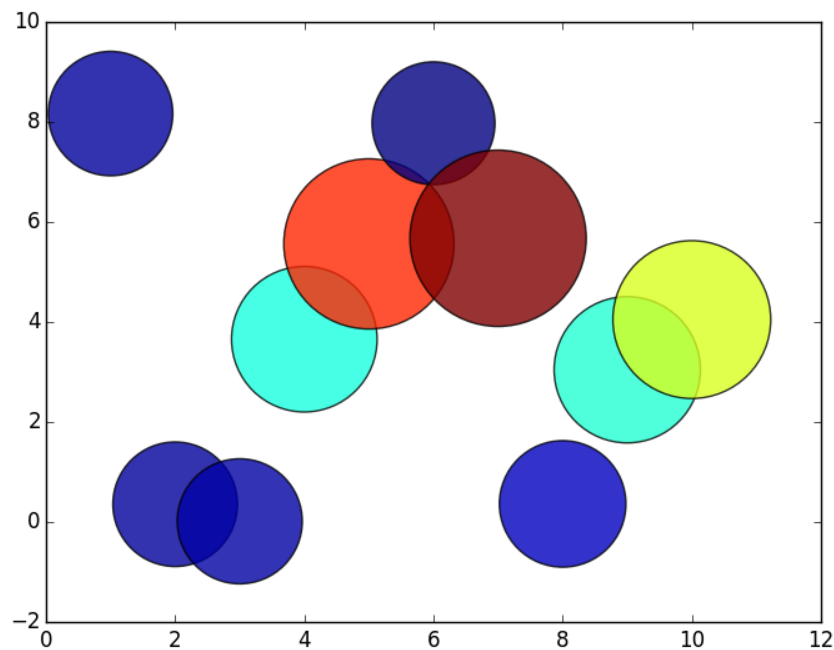
2.3 Scatter plots with scatter Function

Scatter plots may be generated using the `plt.scatter` function. These plots should be given three arguments `x`, `y`, `s`; these are arrays indicating the `x` and `y` values of the data and the size of each data point. The `c` and `alpha` keyword arguments can be used to modify color and opacity of data points. Colors are automatically picked if an array of values is given. Documentation available at

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist

This plots data with `x`-values from 0 to 10, randomly generated `y`-values between 0 and 10, and sizes between 1200 and 30 with some variation. The color scheme is automatically picked according to size.

```
1 >>> import random
2 >>> import matplotlib.pyplot as plt
3 >>> x = [x + 1 for x in range(0, 10)]
4 >>> y = [(elem + random.random() * 10) % 10 for elem in x]
5 >>> sizes = [200 * elem + random.random() * 9300 for elem in x]
6 >>> plt.scatter(x, y, s=sizes, c=sizes, alpha=0.8)
7 >>> plt.show()
```



2.4 Bar Plots with the bar Function

Matplotlib can draw a plot of rectangles, each one bounded by the four edges: left, left + width, bottom, and bottom + height. Here's a demonstration using the color and edgecolor keyword arguments. Documentation is here:

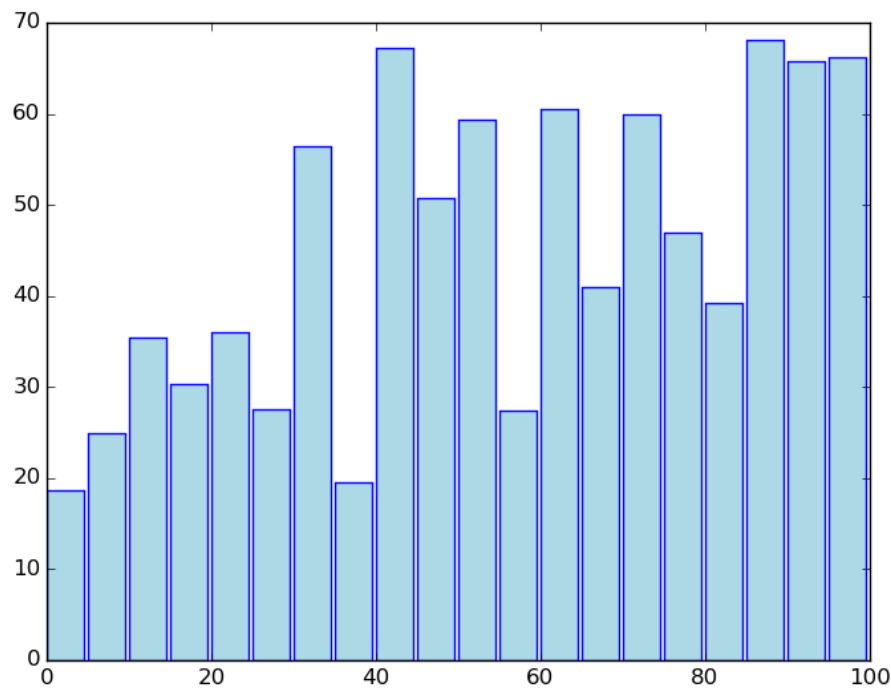
http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar

The rectangles have randomly generated heights and the same width and space between them.

```

1 >>> import random
2 >>> import matplotlib.pyplot as plt
3 >>> lefts = [x for x in range(0,100,5)]
4 >>> print(lefts)
5 [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
6 >>> heights = [x + random.random()*50 + 10 for x in range(0,20)]
7 >>> print(heights)
8 [49.714391342616054, 19.172956449982053, 26.30296873725814, 35.617456459379966,
9  41.00284816312063, 36.42499187921262, 20.105984017686016, 34.56418052389644,
10 24.48833113475763, 54.51007088879054, 51.26910728592864, 27.689576280094144,
11 67.57824233176122, 41.335596453966545, 47.96576291980089, 45.292422625538435,
12 34.460458928440374, 31.992222247836935, 65.24156104706233, 66.47014183737392]
13 >>> plt.bar(left=lefts, height=heights, width=4.5, bottom=0,
14 ...         color='lightblue', edgecolor='blue')
15 >>> plt.show()

```

2.5 Drawing Many Plots with the subplot Function

If you want to draw many plots in the same figure, use the `plt.subplot` function to split the figure into several rows and columns. The first and second arguments are the number of rows and columns. The third argument is the figure number, this should be between 1 and `rows * columns`. Documentation available at

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot

Here we draw 4 plots, three `plt.plots` and one `plt.hist` at the bottom corner.

```

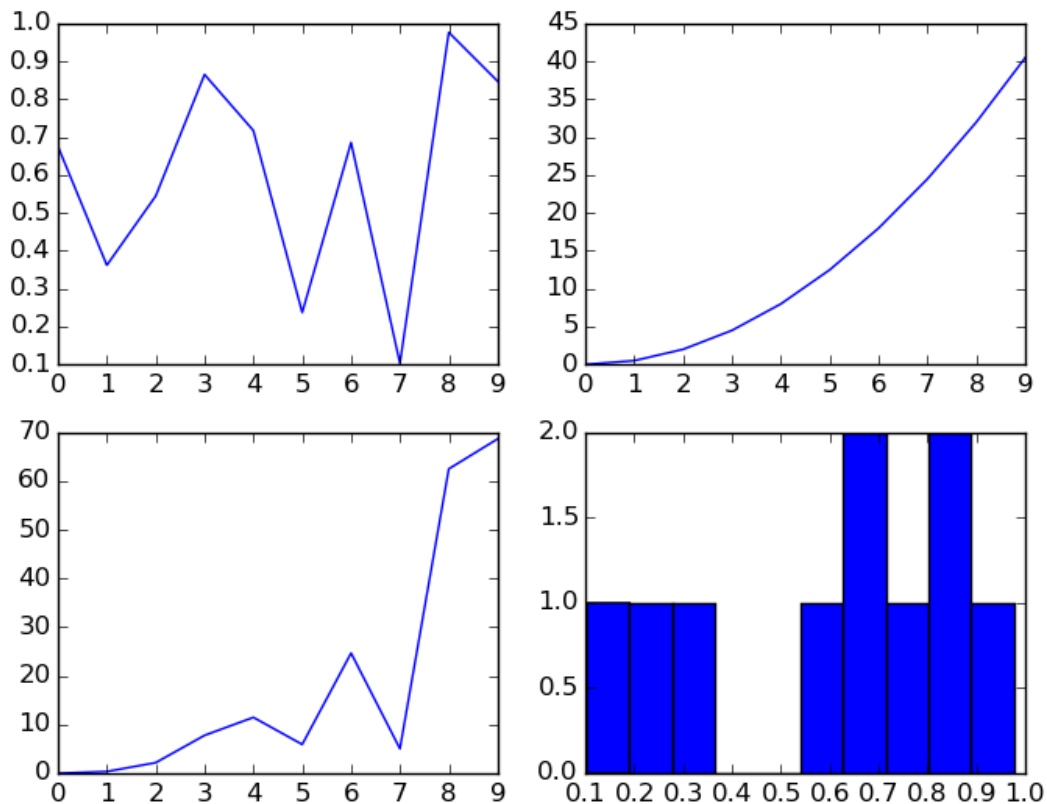
1  import random
2  import matplotlib.pyplot as plt
3
4  x = [elem for elem in range(0,10)]
5  y = [random.random() for elem in x]
6
7  plt.subplot(2, 2, 1)
8  plt.plot(x, y)
9
10 plt.subplot(2, 2, 2)
11 #plt.plot(x, 0.5 * x ** 2)
12 plt.plot(x, [elem ** 2 * 0.5 for elem in x])
13
14 plt.subplot(2, 2, 3)
15 #plt.plot(x, y * x ** 2)

```

```

16 plt.plot(x, [elem * index ** 2 for index, elem in enumerate(y)])
17
18 plt.subplot(2, 2, 4)
19 plt.hist(y)
20
21 plt.show()

```



2.6 Describing your Plots

See these links for documentation:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xlabel

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.ylabel

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.title

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot

Plots can be labeled and titled using the `plt.xlabel`, `plt.ylabel`, and `plt.title` functions. Here's an example:

```

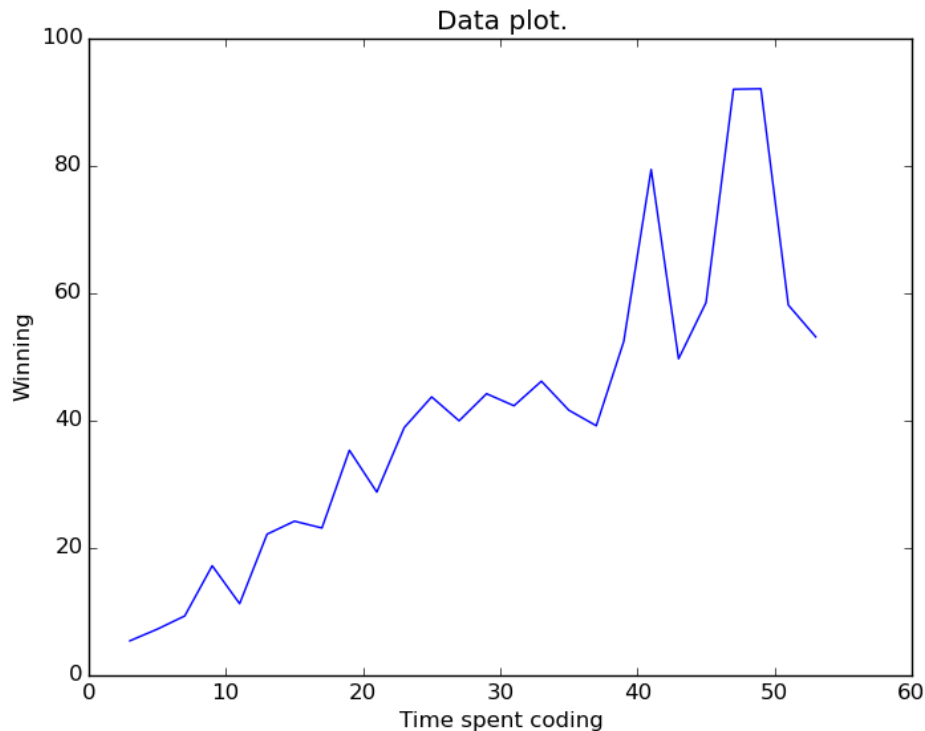
1 import random
2 import matplotlib.pyplot as plt

```

```

3 x = [x + 3 for x in range(0,52,2)]
4 y = [random.random() * x + x for x in x]
5
6 plt.plot(x, y)
7 plt.xlabel('Time spent coding')
8 plt.ylabel('Winning')
9 plt.title('Data plot.')
10 plt.show()

```



What if you want to call attention to an individual data point? Then you should consider the `plt.annotate` function to add annotations to plots. In this example we indicate the maximum y-value in a randomly generated set of data: This code produces a labeled, annotated plot.

Annotation text is set with the `s` argument. The arrow is styled using the `arrowprops` argument. The `xy` and `xytext` keyword arguments are used to place the arrow and text.

```

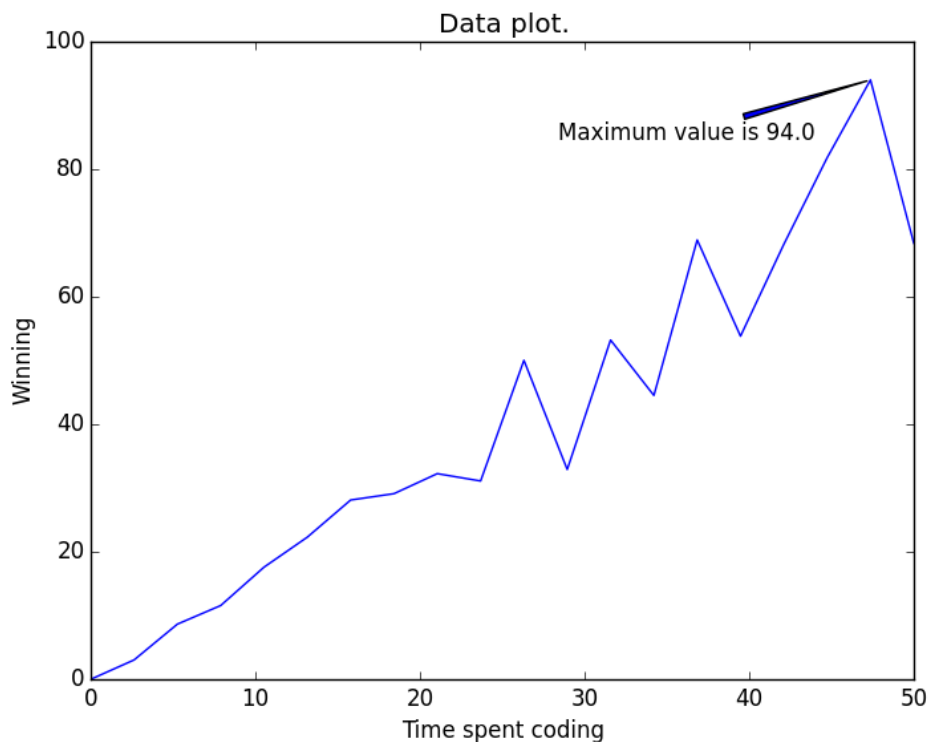
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.linspace(0, 50, num=20)
4 y = np.random.random(20) * x + x
5
6 # find maximum
7 maximum_index = np.argmax(y)
8 maximum = y[maximum_index]
9
10 plt.plot(x, y)

```

```

11 plt.xlabel('Time spent coding')
12 plt.ylabel('Winning')
13 plt.title('Data plot.')
14
15 # add annotation at maximum point
16 plt.annotate(s='Maximum value is {:.3}'.format(y[maximum_index]),
17             xy=(x[maximum_index], y[maximum_index]),
18             xytext=(x[maximum_index] * 0.6, y[maximum_index] * 0.9),
19             arrowprops={'arrowstyle': "wedge,tail_width=0.25"})
20
21 plt.show()

```

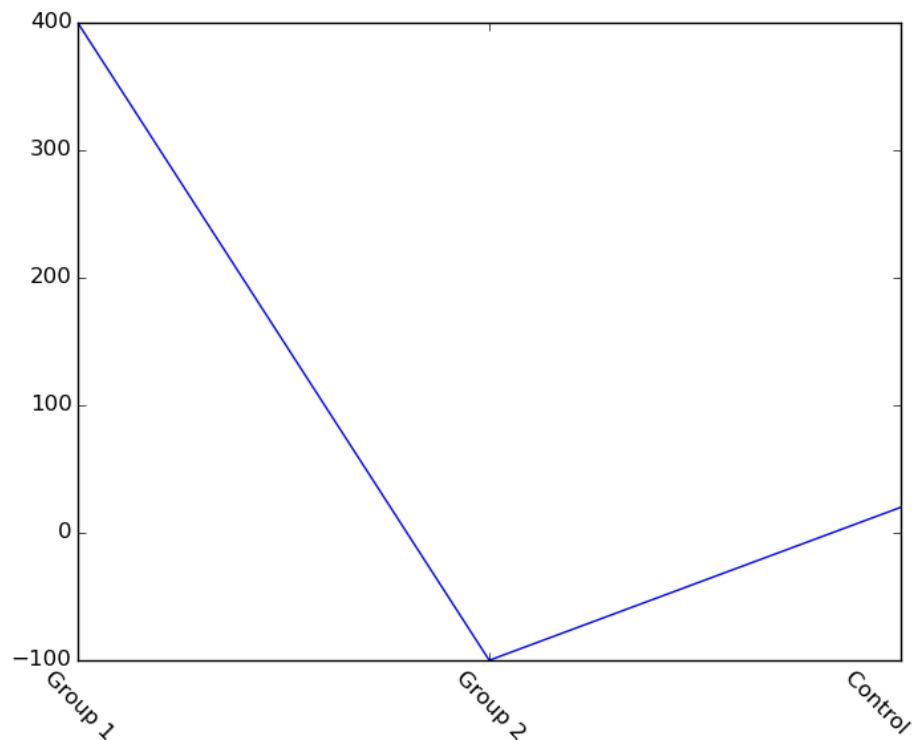


The tick labels can be changed using the `plt.xticks` and `plt.yticks` functions. The first argument is a list of numbers and the second argument is the list of labels for each value. For example,

```

1 >>> plt.plot([1, 2, 3], [400, -100, 20])
2 >>> plt.xticks([1, 2, 3], ["Group 1", "Group 2", "Control"], rotation=-45)
3 >>> plt.show()

```



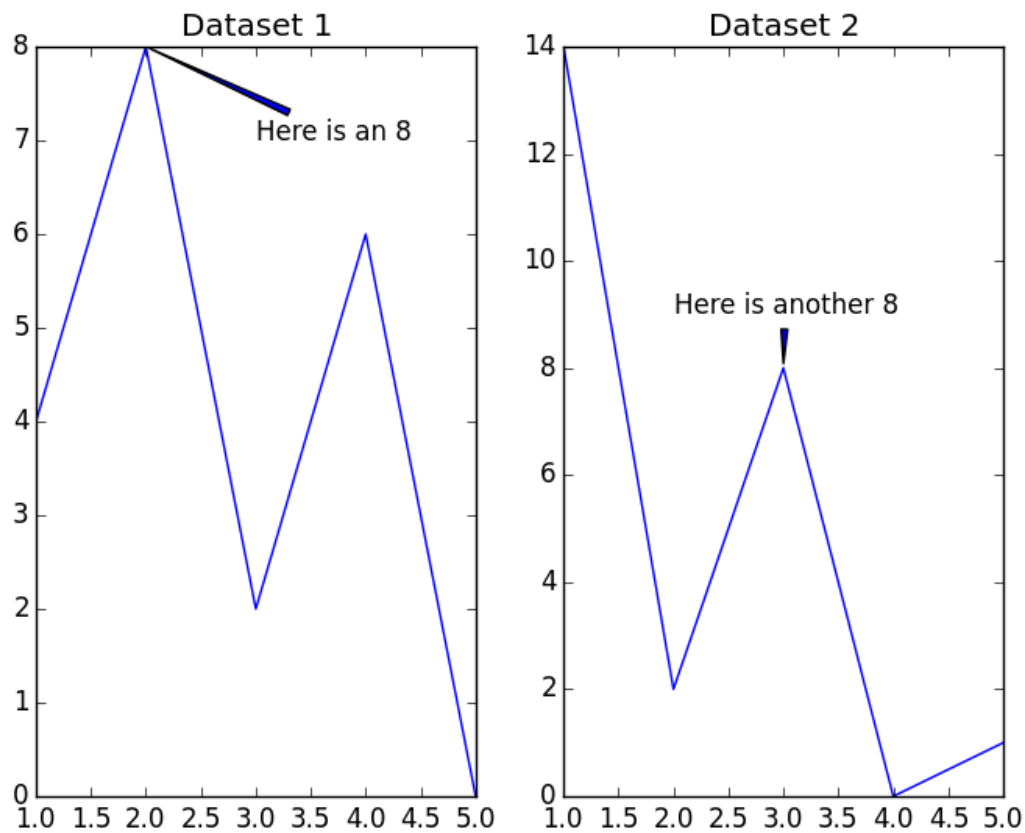
2.7 Subplots and Annotations

Here's an example where subplots have annotations and titles.

```

1 import matplotlib.pyplot as plt
2 x = [1, 2, 3, 4, 5]
3 y1 = [4, 8, 2, 6, 0]
4 y2 = [14, 2, 8, 0, 1]
5
6 plt.subplot(1, 2, 1)
7 plt.plot(x, y1)
8 plt.annotate(s='Here is an 8', xy=(2, 8), xytext=(3,7),
9             arrowprops={'arrowstyle': "wedge,tail_width=0.25"})
10 plt.title('Dataset 1')
11
12 plt.subplot(1, 2, 2)
13 plt.plot(x, y2)
14 plt.annotate(s='Here is another 8', xy=(3, 8), xytext=(2,9),
15             arrowprops={'arrowstyle': "wedge,tail_width=0.25"})
16 plt.title('Dataset 2')
17
18 plt.show()

```



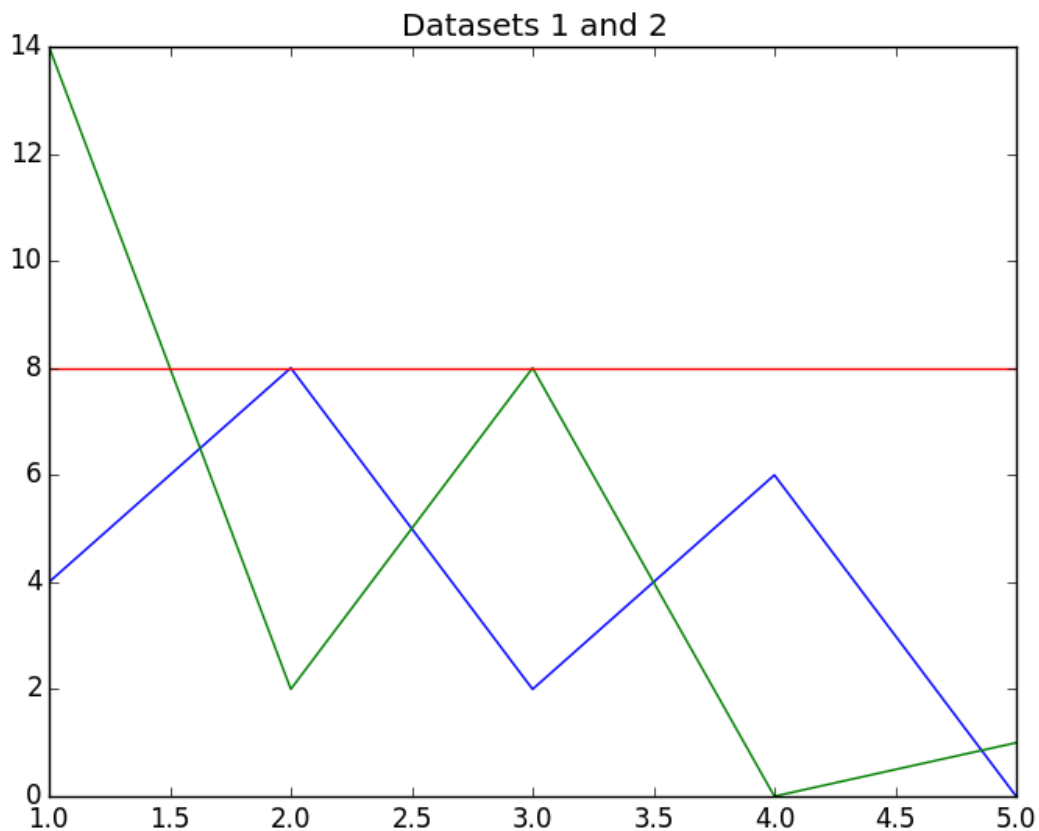
2.8 Overlaying Plots

Plots may be overlaid by calling plot functions multiple times. For example, here are plots of two data sets on the same figure. There is also a line drawn at $y=8$.

```

1 import matplotlib.pyplot as plt
2 x = [1, 2, 3, 4, 5]
3 y1 = [4, 8, 2, 6, 0]
4 y2 = [14, 2, 8, 0, 1]
5
6 plt.plot(x, y1)
7 plt.plot(x, y2)
8 plt.plot([1,5], [8,8])
9 plt.title('Datasets 1 and 2')
10
11 plt.show()

```



2.9 Saving Plots to a File Using `plt.savefig`

To save a plot to a file, replace `plt.show()` with a call to the `savefig` function:

```
1 plt.plot([1, 2, 3], [4, 5, 6])
2 plt.title('Silly plot')
3 plt.savefig('plot.png')
```

You can save to different formats by changing the extension of the filename. For example, the function would produce a JPG file when given a parameter "filename.jpg".

3 Exploring APIs

An important part of this lab is reading module documentation. You have already seen one third party module (Matplotlib), and built-in modules such as `math`. Almost every good Python module has good documentation and if you want to learn how to use one you should start by reading its documentation. For example, Matplotlib supports animation, image rendering, stream plots, error plots, contour plots, log scaled axis, and many other features. You can see examples and detailed specifications in the Matplotlib docs.

3.1 Other Libraries

Other powerful, third-party and well-documented Python modules include 1. *Scrapy*: a web crawling library. 2. *Newspaper*: an Easy to use natural language processor and news downloader. 3. *astropy*: astronomy and astrophysics packages. 4. *vtk*: highly functional 3D visualization library. 5. *pillow*: the friendly fork of the Python Imaging Library. 6. *OpenCV*: a computer vision library. 7. *Django*: used to deliver and manage dynamic websites; powers many large websites.

Visit their websites for information on usage and installation. Here's a list of more interesting libraries:

<https://github.com/vinta/awesome-python>

4 Exercises

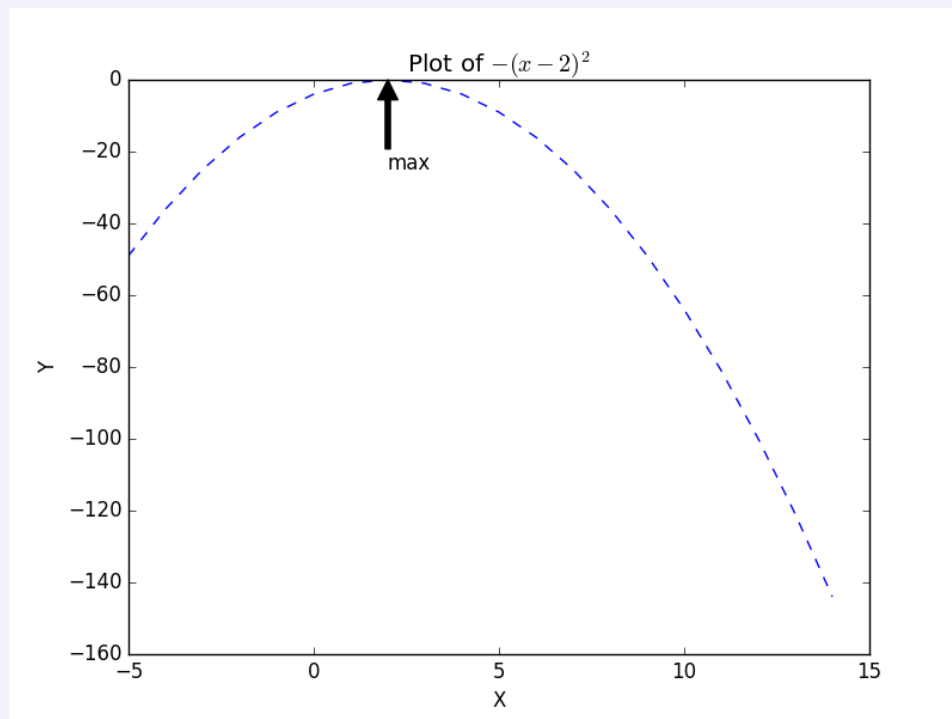
Labels

Remember to label your axes using `plt.xticks()` and `plt.yticks()` and title your graph using `plt.title()`. Read the documentation to find out more.

Exercise 4.1 (plotpoints.py).

Plot the following points. Add labels, a title, and an annotation to the plot as shown in the example. Make sure to draw a dashed blue line. The annotation contains the text “max” and is located at the point (2, 20). The annotation text must be slightly below and the arrow connecting the text and point must be black and have width set to 2.

```
1 xs = range(-5, 15)
2 ys = [-(x - 2) ** 2 for x in xs]
3
4 ...
5
6 plt.title('Plot of  $-(x - 2)^2$ ')
```

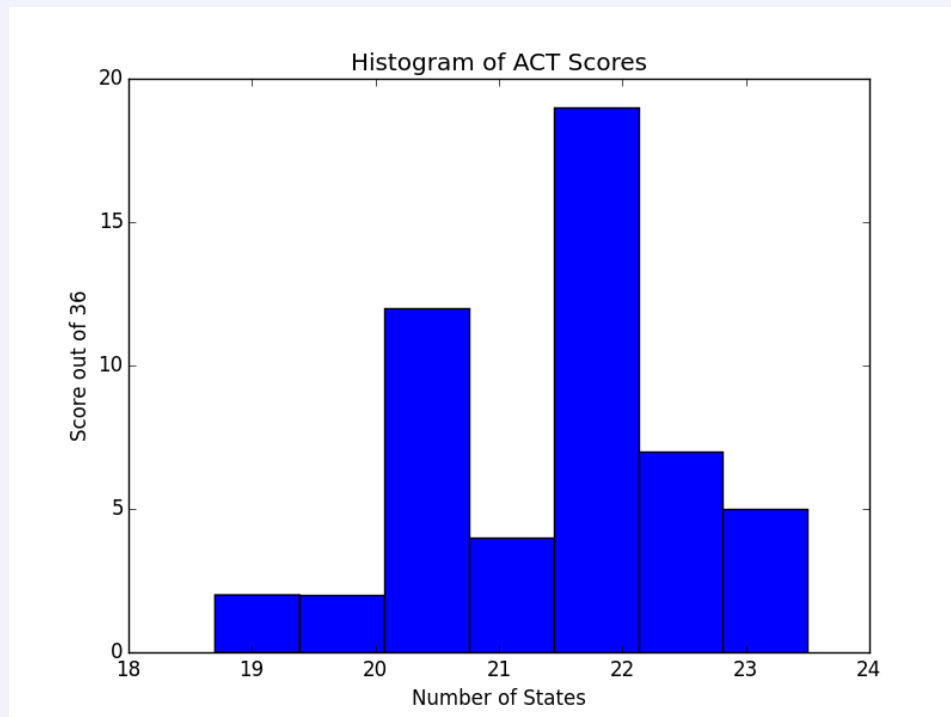


You may use `arrowprops={'color': 'black', 'width': 2}` when drawing the annotation.

Exercise 4.2 (`scorehist.py`).

This exercise relies on the `readscores.py` and `actsat.txt` files from the File I/O lab. You may also rewrite that code using the CSV module.

Write a program that generates a histogram of average ACT scores. For example:

**Exercise 4.3** (`scorebar.py`).

This exercise relies on the `readscores.py` and `actsat.txt` files from the File I/O lab. You may also rewrite that code using the CSV module.

Write a program that draw the following charts.

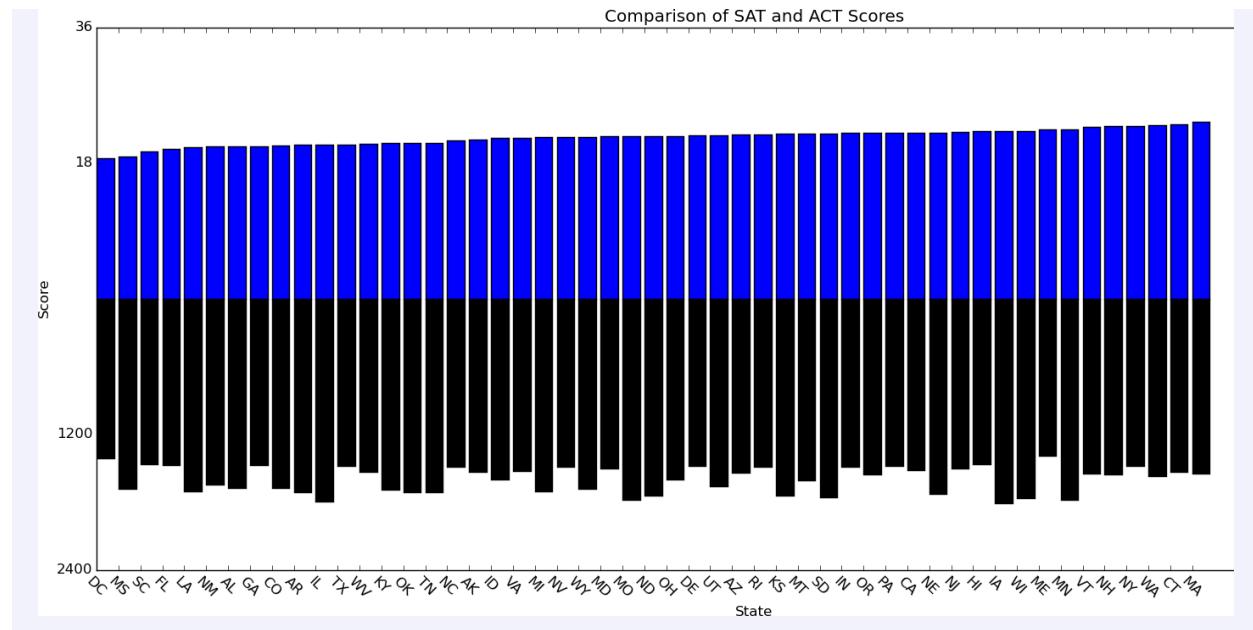
1. A bar chart where each state/territory is represented by two bars, one for the composite ACT score and the other for the total score of all 3 SAT tests.

Each SAT section is out of 800 points, while the ACT is out of 36 points. Each bar should have the same maximum heights, but the labels should have the correct SAT or ACT score. See the example image.

2. Produce the same chart as in part 2, but only for states in which less than 50% take the ACT and more than 50% take the SAT. (There should be 21 states/territories like this.)

Hint: Notice the scale is different for each plot. One way of adjusting the scales is to divide each number in the data so that every number is between 0 and 1. For the SAT scores, a score of 0 would be 0 and a score of 2400 would be 1.0. Then assign the label "0" to 0, and the label "1800" to 0.5, and the label "2400" to 1.0. Similarly, -1 would have label "36".

This is an example of the first bar plot:



Note

None of these plots have to be exactly the same as the examples, they need only convey the same information in an understandable plot.

5 The CSV Module

CSV files are used to store tabular data. They have comma separated values on each line. Most spreadsheet programs support CSV output and input. Python comes with the CSV module for reading and writing such files. It supports any delimiter so you can also manage space-separated and tab-separated data. To use the module, create a CSV reader using the filename of the data file. For example, suppose we have the file `short.csv`. It has column labels on the first row and columns are separated by commas.

```
1 Entity,x,y,z
2 Fruit tree,1,2,3
3 Bats, 1, 3, 10
```

We can use the `csv.DictReader` function to parse this data. It takes an open file as the first argument and returns something that you can use a for-loop to traverse. It essentially returns a list of dictionaries. For example,

```
1 >>> import csv
2 >>> with open('short.csv') as f:
3 ...     l = [e for e in csv.DictReader(f)]
4 >>> print(l)
5 [{'Entity': 'Fruit tree', 'x': '1', 'y': '2', 'z': '3'},
6  {'Entity': 'Bats', 'x': '1', 'y': '3', 'z': '10'}]
```

Read the documentation to learn about writing CSV files and the `csv.Sniffer`, which can be used to deduce the format of a CSV file.

<https://docs.python.org/3/library/csv.html>

The data from <http://www.gapminder.org/data/> was converted to CSV and parsed using Python's CSV module to form the file `country_data.py`.

Extra Credit (10 Points)

Use matplotlib to draw scatter plots relating worldwide life expectancy, income per person, and population for the years 2010 and 1960.

You may access the data by completing the extra credit or by importing `country_data.py`. The data is stored in the `data1960` and `data2010` variables and looks like:

```
1 data1960 = [{'country': 'Andorra',
2             'expectancy': None,
3             'gdp': 15234.0,
4             'population': 13414.0},
5             {'country': 'Eritrea',
6             'expectancy': 41.8278,
7             'gdp': 885.0,
8             'population': 1407631.0}, ...
```

See “200 Years, 200 Countries” by Hans Rosling if you want a good presentation on this data. Notice the special value `None` is present. To check if a variable `x` is `None`, type `x == None` or `x is None`.

You are required to:

1. Draw a scatter plot with `x` and `y` coordinates set to income and life expectancy. If there is a `None` value, replace it with the minimum income or the minimum life expectancy or an area set to zero.
2. The area of each data point should be related to the population of every country. You can choose how to scale the data, but to make it look like the example, you may use:

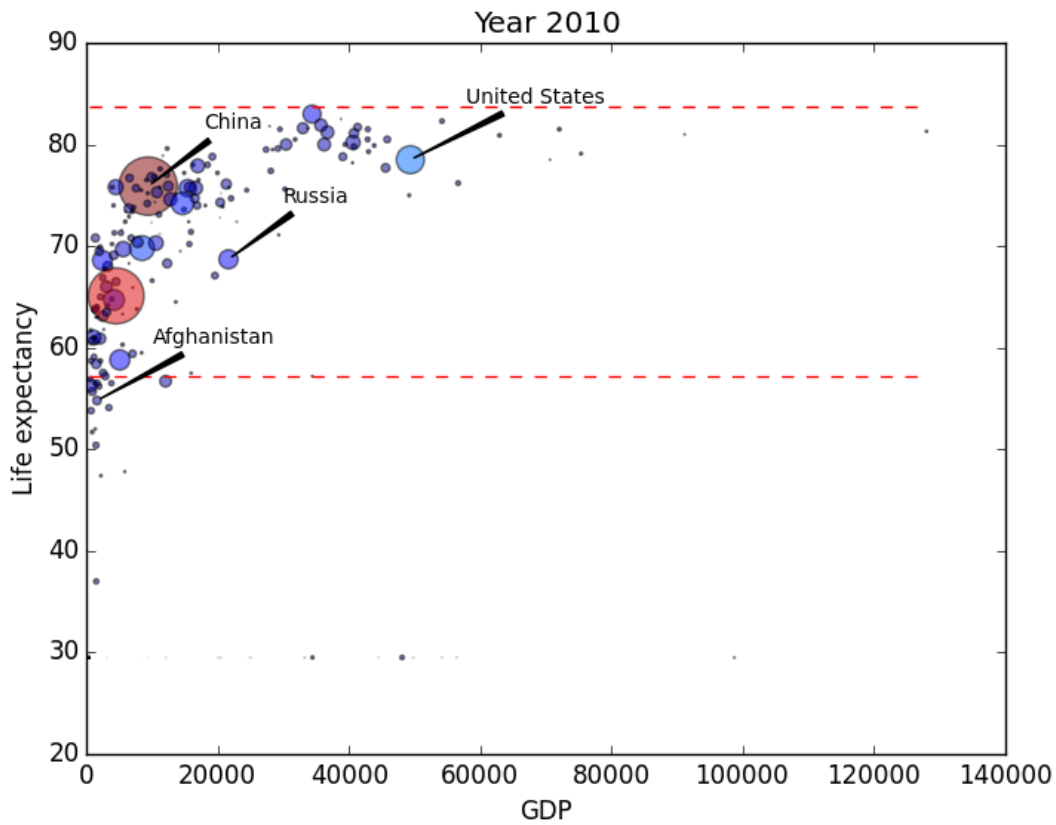
```
1 scaled_pop = [800 * p / max(pop) for p in pop]
```

3. The scatter plots must be labeled and titled. X label: “Income”, y label: “Life expectancy”, title: “Year (year of data)”.
4. Annotate China, Russia, United States, and Afghanistan as shown. Annotation text should be drawn away from the annotation coordinates. You can get an arrow like in the example with this keyword argument:

```
1 arrowprops={'arrowstyle': 'wedge,tail_width=0.25', 'color': 'black'}
```

5. Draw dashed red lines at the mean and at the max life expectancy. Python's max function is useful for this.

Here is the plot for the 2010 and 1960 data:



Supplementary Files

The `country_data.py` file is available on canvas.

Submitting

You should submit your code as a tarball that contains all the exercise files for this lab. The submitted file should be named

`cse107_firstname_lastname_lab11.tar.gz`

Upload your tarball to Canvas.

List of Files to Submit

4.1	Exercise (plotpoints.py)	15
4.2	Exercise (scorehist.py)	16
4.3	Exercise (scorebar.py)	16

Exercises start on page 15.